BIOINFORMATICS APPLICATIONS NOTE
Vol. 25 no. 8 2009, pages 1084–1085
doi:10.1093/bioinformatics/btp112

Sequence analysis

mkESA: enhanced suffix array construction tool

Robert Homann[1,2,*], David Fleer[2], Robert Giegerich[2] and Marc Rehmsmeier[3,†]

[1]International NRW Graduate School in Bioinformatics and Genome Research, Center for Biotechnology (CeBiTec), Bielefeld University, 33594 Bielefeld, [2]Technische Fakultät, Bielefeld University, Postfach 100 131, 33501, Bielefeld, Germany and [3]GMI - Gregor Mendel Institute of Molecular Plant Biology GmbH, Dr. Bohr-Gasse 3, 1030 Vienna, Austria

Received on January 21, 2009; revised on February 19, 2009; accepted on February 20, 2009

Advance Access publication February 26, 2009

Associate Editor: Limsoon Wong

ABSTRACT

Summary: We introduce the tool *mkESA*, an open source program for constructing enhanced suffix arrays (ESAs), striving for low memory consumption, yet high practical speed. *mkESA* is a user-friendly program written in portable C99, based on a parallelized version of the Deep-Shallow suffix array construction algorithm, which is known for its high speed and small memory usage. The tool handles large FASTA files with multiple sequences, and computes suffix arrays and various additional tables, such as the LCP table (longest common prefix) or the inverse suffix array, from given sequence data.

Availability: The source code of *mkESA* is freely available under the terms of the GNU General Public License (GPL) version 2 at http://bibiserv.techfak.uni-bielefeld.de/mkesa/.

Contact: rhomann@techfak.uni-bielefeld.de

1 INTRODUCTION

The program *mkESA* is a software tool for constructing enhanced suffix arrays (ESAs) from biological sequence data. The ESA is an index data structure for textual data, introduced in Abouelhoda *et al.* (2004) as an extension of the well-known suffix array (Manber and Myers, 1993). The ESA is equivalent to the suffix tree, another very important, but more space consuming full-text index data structure (Gusfield, 1997). The major advantages of ESAs over suffix trees are their lower space overhead, improved locality of reference and simple storing to files.

A suffix array for text T of length n is a table of size $n+1$ that lists the start positions of the suffixes of T in lexicographic order. Using a suffix array, exact string queries can be answered in $O(m\log n)$ time, where m is the length of the query, instead of $O(m+n)$ time without a suffix array. ESAs are composed of a suffix array and additional tables that can be used to improve query performance [e.g. $O(m+\log n)$ time using the LCP table, called *Hgt* array in Manber and Myers (1993)], or enabling efficient implementation of more advanced queries (e.g. finding maximum unique matches). Thus, ESAs are fundamental technology in sequence analysis.

Many interesting problems on sequences from the field of computational biology can be solved efficiently by transforming sequence data into (enhanced) suffix arrays [see, for instance, (Beckstette *et al.*, 2006; De Bona *et al.*, 2008; Höhl *et al.*, 2002; Krumsiek *et al.*, 2007; Rahmann, 2003)]. Linear-time algorithms for suffix array construction have been proposed as well as algorithms that are fast in practice and/or tuned for space efficiency, rendering use of suffix arrays feasible for large datasets; see Puglisi *et al.* (2007) for a comprehensive overview. In addition, by the results of Abouelhoda *et al.* (2004), any program using suffix trees can be transformed so to employ ESAs instead and benefit from the advantages offered by that data structure.

Despite the great interest in suffix arrays in the literature, only few actual programs for ESA construction are available. Most existing programs are useful for mere suffix array construction, and do not address specificities of computational biology such as handling multiple sequences and very large datasets. A notable exception is the widely used *mkvtree* program (http://www.vmatch.de/). *mkvtree* can read common file formats such as FASTA and keeps sequences separated from their descriptions. An ESA generated by *mkvtree* may contain multiple sequences, stored so that a match can easily be mapped to its corresponding sequence. The program is available free of charge as part of the *Vmatch* package, but, unfortunately, in binary form and for non-commercial purposes only. This implies that software relying on *mkvtree* cannot be distributed easily since the terms of the *Vmatch* license agreement restrict the legal use of *mkvtree*. Software that requires using *mkvtree* also requires all users to obtain the *Vmatch* package, if available for their platform of choice, and have them sign a license agreement, too.

We have implemented the alternative open source software tool *mkESA*, using the Deep-Shallow algorithm (Manzini and Ferragina, 2004) for in-memory suffix array construction instead of multikey quicksort as used by *mkvtree*. Thus, *mkESA* is efficient even for highly repetitive sequence data, and is fast as long as all data can be held in main memory. As further improvement, our implementation of Deep-Shallow can use multiple CPUs for increased speed.

2 IMPLEMENTATION

With *mkvtree* being the most widely spread program for ESA construction, we tried to pick up all of the important ideas implemented in *mkvtree* and improve upon its weaknesses. *mkESA* has been designed so to produce output as compatible with *mkvtree* as possible. The files generated by *mkESA* are in fact the same as those made by *mkvtree*, meaning that data produced by *mkESA* can be processed by programs that expect *mkvtree*-generated ESAs.

*To whom correspondence should be addressed.

†Present address: GMI - Gregor Mendel Institute of Molecular Plant Biology GmbH, Dr. Bohr-Gasse 3, 1030 Vienna, Austria.

Table 1. Datasets used for performance measurements

Name	Description	Size	σ
chr1	Chromosome 1 human genome	219 (219) MB	4
fmdv	Foot/mouth disease virus genomes	65 (64) MB	4
spro	UniprotKB/Swiss-Prot rel. 56.4	181 (140) MB	20
trem	UniprotKB/TrEMBL rel. 39.4	2836 (2110) MB	20
f25	25th Fibonacci string	73 (73) kB	2
f30	30th Fibonacci string	813 (813) kB	2

Sizes are given as file sizes, followed by sizes of files with FASTA headers removed in parentheses. Alphabet sizes are given as σ. We included Fibonacci strings since these are hard on many suffix tree and suffix array construction algorithms due to their high repetitiveness. They impose the worst case for the number of nodes in a suffix tree, $2n$, and thus, e.g. trigger the worst case running time of $O(n^2)$ of the WOTD suffix tree construction algorithm (Giegerich *et al.*, 2003). Dataset 'fmdv' is a non-artificial example for highly repetitive sequence data, with similar impact on performance (Table 2).

mkESA employs the 'Deep-Shallow' algorithm of Manzini and Ferragina (2004) for suffix array construction. This algorithm belongs to the family of 'lightweight' suffix sorting algorithms, covering algorithms that use only very small additional space besides the suffix array and the input text, i.e. only $O((5+\epsilon)n)$ bytes space for a text of length n, and using 32 bit integers for the suffix array. Our version of Deep-Shallow is multithreaded, i.e. the computational work for suffix sorting can be distributed over multiple CPUs or CPU cores. Since Deep-Shallow is not useful for building LCP tables as by-product of suffix sorting (as is the case with simple multikey quicksort), we use the space-efficient, linear-time algorithm of Manzini (2004) to construct LCP tables from suffix arrays. Moreover, *mkESA* can generate the inverse suffix array and the skip table (Beckstette *et al.*, 2006). It is worth noting that *mkESA* can incrementally add additional tables when they are needed.

3 RUNTIME BENCHMARKS

We compared the performance of *mkESA* with other programs for suffix array construction, namely *mkvtree*, *mksary* 1.2.0 (http://sary.sourceforge.net/, included for its ability to run multithreaded), and Manzini's implementation of Deep-Shallow *ds*. We measured the time and space consumption for building suffix arrays from the datasets in Table 1, using `memtime` version 1.3. *mkESA* and *mkvtree* processed FASTA files, the other programs processed the bare sequence data with FASTA headers removed so that all programs had comparable workloads. Only 'parallel *mkESA*' and 'parallel *mksary*' (Table 2) made explicit use of multiple CPU cores. Measurements were taken on a Sun Fire X4450 (4 Intel Xeon CPUs at 2.93 GHz, 16 cores, 96 GB RAM) running Solaris 10. The programs were compiled with *gcc* 4.1.1 using flags `-m64 -O3 -fomit-frame-pointer`. Each experiment was repeated four times in a row; the best (shortest elapsed time) of the results are displayed in Table 2. Our results show comparable memory requirements for all tested programs, while *mkESA* is usually the fastest among them, even when using only one CPU.

4 CONCLUSION

We presented *mkESA*, a portable, lightweight, multithreaded and fast program for constructing enhanced suffix arrays. We carefully tested the software on a variety of UNIX-like operating systems and hardware architectures, including recent versions of Linux, Solaris, Mac OS X, FreeBSD, OpenBSD and NetBSD. Its ability to generate output compatible with *mkvtree* makes *mkESA* a convenient open source drop-in replacement for earlier programs.

Table 2. Results of performance measurements

Name	*mkESA*		Parallel *mkESA*		*mkvtree*	
	sec	MB	sec	MB	sec	MB
chr1	91 (2.6)	1085	66 (2.6)	1093	138 (2.2)	1148
fmdv	89 (0.9)	353	66 (0.9)	356	1797 (1.1)	338
spro	47 (1.9)	785	25 (1.9)	785	76 (2.2)	813
trem	2273 (545)	21 461	1500 (553)	21 462	2956 (530)	21 827
f25	0.1 (0.0)	0.1	0.1 (0.0)	0.1	7.3 (0.0)	1.4
f30	1.1 (0.0)	5.1	1.1 (0.0)	5.3	895 (0.0)	5.4
Name	*mksary*		Parallel *mksary*		*ds*	
	sec	MB	sec	MB	sec	MB
chr1	224 (11)	1097	252 (28)	1097	102 (3.8)	1098
fmdv	–	–	–	–	99 (1.2)	323
spro	161 (7.7)	705	115 (23)	707	63 (2.5)	705
f25	7.5 (0.0)	3.2	6.3 (0.0)	3.4	0.1 (0.0)	0.1
f30	–	–	–	–	0.9 (0.0)	5.1

The 'sec' columns show the total time consumed in seconds (wall time clock), followed by the time attributed to operating system activities in parentheses. The 'MB' columns show main memory consumption in megabytes [resident set size (RSS)]. Parallel versions were allowed to use up to 16 threads. Some programs crashed for various datasets, in which cases results are not shown. For the same reason there is no row for 'trem' in the lower part. All values were rounded for readability.

Conflict of Interest: none declared.

REFERENCES

Abouelhoda,M. *et al*. (2004) Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algorithms*, **2**, 53–86.

Beckstette,M. *et al*. (2006) Fast index based algorithms and software for matching position specific scoring matrices. *BMC Bioinformatics*, **7**.

De Bona,F. *et al*. (2008) Optimal spliced alignments of short sequence reads. *Bioinformatics*, **24**, i174–i180.

Giegerich,R. *et al*. (2003) Efficient implementation of lazy suffix trees. *Softw. Pract. Exp.*, **33**, 1035–1049.

Gusfield,D. (1997) *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA.

Höhl,M. *et al*. (2002) Efficient multiple genome alignment. *Bioinformatics*, **18**, S312–S320.

Krumsiek,J. *et al*. (2007) Gepard: a rapid and sensitive tool for creating dotplots on genome scale. *Bioinformatics*, **23**, 1026–1028.

Manber,U. and Myers,E. (1993) Suffix Arrays: a new method for on-line string searches. *SIAM J. Comput.*, **22**, 935–948.

Manzini,G. (2004) Two space saving tricks for linear time LCP array computation. In Hagerup,T. and Katajainen,J. eds, *Proceedings of 9th Scandinavian Workshop on Algorithm Theory (SWAT '04)*, Vol. 3111 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, pp. 372–383.

Manzini,G. and Ferragina,P. (2004) Engineering a lightweight suffix array construction algorithm. *Algorithmica*, **40**, 33–50.

Puglisi,S.J. *et al*. (2007) A taxonomy of suffix array construction algorithms. *ACM Comput. Surv.*, **39**, 4.

Rahmann,S. (2003) Fast large scale oligonucleotide selection using the longest common factor approach. *J. Bioinform. Comput. Biol.*, **1**, 343–361.